37

**Data Structures** are a specialized means of organizing and storing data in computers in such a way that we can perform operations on the stored data more efficiently. Data structures have a wide and diverse scope of usage across the fields of Computer Science and Software Engineering.
Data structures are being used in almost every program or software system that has been developed. Moreover, data structures come under the fundamentals of Computer Science and Software Engineering. It is a key topic when it comes to Software Engineering interview questions. Hence as developers, we must have good knowledge about data structures.
An **array** is a structure of fixed-size, which can hold items of the same data type. It can be an array of integers, an array of floating-point numbers, an array of strings or even an array of arrays (such as *2-dimensional arrays*). Arrays are indexed, meaning that random access is possible.

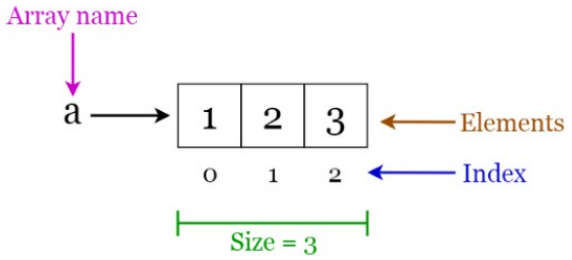


Fig 1. Visualization of basic Terminology of Arrays (Image by author)

### Array operations

- **Traverse**: Go through the elements and print them.
- **Search**: Search for an element in the array. You can search the element by its value or its index
- **Update**: Update the value of an existing element at a given index

**Inserting** elements to an array and **deleting** elements from an array cannot be done straight away as arrays are fixed in size. If you want to insert an element to an array, first you will have to create a new array with increased size (current size + 1), copy the existing elements and add the new element. The same goes for the deletion with a new array of reduced size.

### Applications of arrays

- Used as the building blocks to build other data structures such as array lists, heaps, hash tables, vectors and matrices.
- Used for different sorting algorithms such as insertion sort, quick sort, bubble sort and merge sort.

38

A **linked list** is a sequential structure that consists of a sequence of items in linear order which are linked to each other. Hence, you have to access data sequentially and random access is not possible. Linked lists provide a simple and flexible representation of dynamic sets.

Let's consider the following terms regarding linked lists. You can get a clear idea by referring to Figure 2.

- Elements in a linked list are known as **nodes**.
- Each node contains a **key** and a pointer to its successor node, known as **next**.
- The attribute named **head** points to the first element of the linked list.
- The last element of the linked list is known as the **tail**.


Fig 2. Visualization of basic Terminology of Linked Lists (Image by author)

Following are the various types of linked lists available.

- **Singly linked list** — Traversal of items can be done in the forward direction only.
- **Doubly linked list** — Traversal of items can be done in both forward and backward directions. Nodes consist of an additional pointer known as **prev**, pointing to the previous node.
- **Circular linked lists** — Linked lists where the prev pointer of the head points to the tail and the next pointer of the tail points to the head.

### Linked list operations

- **Search**: Find the first element with the key **k** in the given linked list by a simple linear search and returns a pointer to this element
- **Insert**: Insert a key to the linked list. An insertion can be done in 3 different ways; insert at the beginning of the list, insert at the end of the list and insert in the middle of the list.
- **Delete**: Removes an element **x** from a given linked list. You cannot delete a node by a single step. A deletion can be done in 3 different ways; delete from the beginning of the list, delete from the end of the list and delete from the middle of the list.

### Applications of linked lists

- Used for *symbol table management* in compiler design.
- Used in switching between programs using Alt + Tab (implemented using Circular Linked List).

Excerpts from Mallawaarachchi, V. (2020). *8 Common Data Structures Every Programmer Must Know*. towardsdatascience.com

39

A **stack** is a **LIFO** (Last In First Out — the element placed at last can be accessed at first) structure which can be commonly found in many programming languages. This structure is named as “stack” because it resembles a real-world stack — a stack of plates.

### Stack operations

Given below are the 2 basic operations that can be performed on a stack. Please refer to Figure 3 to get a better understanding of the stack operations.

- **Push**: Insert an element on to the top of the stack.
- **Pop**: Delete the topmost element and return it.

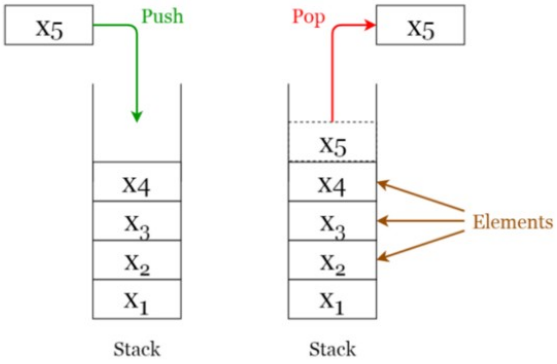


Fig 3. Visualization of basic Operations of Stacks (Image by author)

Furthermore, the following additional functions are provided for a stack in order to check its status.

- **isEmpty**: Check if the stack is empty.
- **isFull**: Check if the stack is full.

### Applications of stacks

- Used for expression evaluation (e.g.: *shunting-yard algorithm* for parsing and evaluating mathematical expressions).
- Used to implement function calls in recursion programming.

40

A **queue** is a **FIFO** (First In First Out — the element placed at first can be accessed at first) structure which can be commonly found in many programming languages. This structure is named as “queue” because it resembles a real-world queue — people waiting in a queue.

### Queue operations

Given below are the 2 basic operations that can be performed on a queue. Please refer to Figure 4 to get a better understanding of the queue operations.

- **Enqueue**: Insert an element to the end of the queue.
- **Dequeue**: Delete the element from the beginning of the queue.

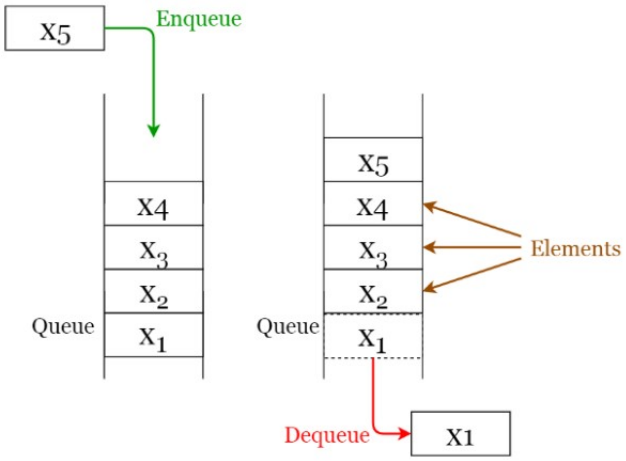


Fig 4. Visualization of Basic Operations of Queues (Image by author)

### Applications of queues

- Used to manage threads in multithreading.
- Used to implement queuing systems (e.g.: priority queues).